8Б

УДК 681.3

М.К. Буза
Белорусский государственный университет, г. Минск, Беларусь
bouza@bsu.by

Технологический подход к проектированию параллельной обработки

В статье предложен концептуальный подход и методика проектирования параллельных программ и метод ускоренной обработки данных.

Введение

Исследования в области информационных наук безграничны по направлениям, тематикам, уровням подходов (от философии до конкретных приложений). Однако сегодня в связи с ураганно-динамичным развитием событий во всех сферах мироздания для обеспечения устойчивого развития жизни на нашей планете (а может, и жизни вообще), в первую очередь, надо исследовать и разрабатывать вопросы, которые могут быть полезны человеку.

Сегодня мир беспокоят различные проблемы, среди которых борьба (а лучше – профилактика) с различными заболеваниями, механизмы работы человеческого мозга (ряд исследователей пришли к заключению, что утверждение: возможности человеческого мозга мы используем только на 10 % – миф), последствия столкновения с кометами и астероидами (возможно, завершение существования жизни на Земле, а может, и планеты вообще) и ряд других.

Решение одной из важнейших проблем, интересующих значительную часть научного мира, – механизм формирования памяти – может иметь существенное влияние на развитие различных областей нашей жизнедеятельности. В частности, решение задачи кодирования памяти открыло бы путь к созданию нового поколения интеллектуальных компьютеров.

Постановка задачи

Попытки формализовать и решить указанные проблемы требуют создания совершенно новых методов и алгоритмов. Необходимы и новые построения в области brainware, hardware, software.

Поиски последних лет привели к серьезным исследованиям и практическим разработкам систем параллельного действия в сфере параллельной обработки данных, вселяющим надежду, что перечисленные проблемы могут быть решены за разумное время. Однако многопроцессорные системы экономически эффективны лишь тогда, когда в обрабатываемой программе параллелизм задан изначально.

Поэтому актуальной становится задача создания методики проектирования параллельных программ и поддерживающего его инструментария, позволяющего в значительной степени автоматизировать этот процесс.

Один из подходов к проектированию параллельной обработки представлен в данной статье.

Целью статьи является создание механизма проектирования параллельных программ и метода обработки данных, которые позволили бы в значительной степени упростить процесс подготовки программ и данных для эффективной работы многопроцессорных и многоядерных обрабатывающих устройств. Появление многоядерных процессоров сделало средства построения параллельных программ актуальными даже для персональных компьютеров.

Применение многопроцессорных систем становится эффективным только при наличии такой программной поддержки их функционирования, которая обеспечила бы максимальную загрузку всех компонент. Важное влияние на общую производительность оказывают и используемые языки программирования. Потенциал применения современных систем (кластеров, метакластеров и GRID) существенно зависит от качества подготовленных программ.

Концептуальный подход

На сегодняшний день начал активно использоваться MC# (Multiprocessor C) – язык программирования, расширяющий стандарт языка C# [1] как одного из интуитивно понятных для использования при проектировании параллельных программ, имеющий свои реализации под Windows и Linux-платформы.

Предлагаемый ниже механизм создания инструментальной системы в значительной степени позволяет для заданной последовательной программы различать типы обрабатываемых фрагментов, характерных для определенного класса алгоритмов, базируется на выделении активных паттернов (т.е. готовых для обслуживания), их активизации и интеграции результатов обработки.

В динамике идет наблюдение за состоянием паттернов активности с тем, чтобы формировать след вычислений конечного результата. Формирование следа дает возможность создавать контрольные точки, позволяющие в случае необходимости активизировать необходимые паттерны для аналогичной обработки, например, в случае обнаруженного сбоя или некорректного формирования интегрального результата. Это сродни наблюдению за процессом воспоминания о прошедших акциях. Количественные измерения спонтанной реактивации паттернов позволяют решать задачу минимизации программных средств и требуемой памяти.

Можно фиксировать различные типы обработки фрагментов исходной программы (линейный участок, ветвление, цикл, обращение к памяти и т.д.), используя для их интерпретации своеобразный многоугольник, число вершин которого равно количеству определяемых типов обработки до наступления состояния покоя. С каждой из его вершин может быть связан некоторый новый многоугольник (например, с вершиной цикла, может быть связан многоугольник, характеризующий тип цикла: простой цикл, с фиксированным повторением тела цикла, итерационный цикл и т.д.), детализирующий процесс обработки.

Таким образом, выстраивается некоторая иерархическая структура (ИС), которую можно проецировать на реальную программу, создавая специализированную (адаптированную) иерархическую структуру, позволяющую сократить время обработки данных, варьируя глубиной вложения.

Научившись формировать ИС, характерную для выбранного класса алгоритмов, можно в значительной степени автоматизировать процесс параллельной обработки данных. Каждая конкретная акция над программой будет формировать конкретную последовательность событий, приводящих к активации необходимых паттернов. Таким образом, всегда информация о необходимой обработке будет представлена ансамблем паттернов, организованных иерархически по принципу от общего к частному.

Иерархический способ формирования структуры обработки может быть использован для различных классов алгоритмов, покрывающих такие специфические предметные области, как биология, правоведение, астрономия и т.д., благодаря созданию неограниченного количества паттернов – от общих до уникальных. Отображение новых акций из конкретной предметной области может осуществляться простой заменой конкретных паттернов в реализующей иерархии.

Структура ИС представлена на рис. 1, где Р – исходная программа. Каждая вершина ИС может интерпретироваться как многоугольник, количество вершин которого определяется многообразием предлагаемой обработки на каждом уровне, а K_i^j – ряд конкретизирующих паттернов, которые могут ветвиться до требуемого уровня.

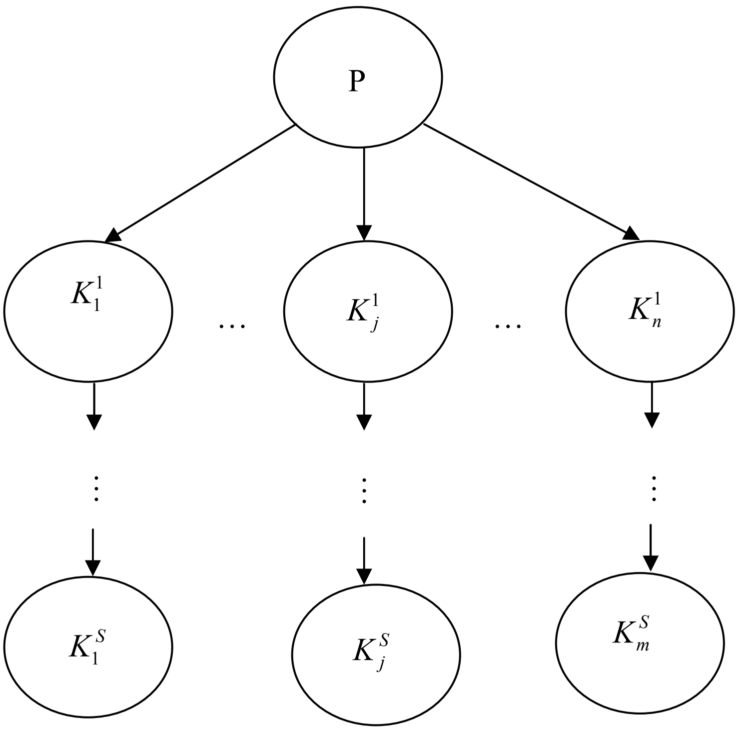


Рисунок 1 – Общая структура ИС

ИС всегда как бы настраивается на конкретную акцию. Действительно, сформировав общее представление о цикле, мы распознаем его в любой программе, несмотря на то, что эту конкретную программу никогда не обрабатывали.

Важно отслеживать, как будут меняться паттерны активности в динамике в зависимости от количества обработчиков, возможностей операционной системы и программного обеспечения вообще.

Методика проектирования параллельной обработки

На сегодняшний день имеются различные средства для проектирования и описания параллельных процессов [2].

Разработаны специальные примитивы *коммуникации* и *управления*, позволяющие программисту создавать сценарии параллельного исполнения программ; имеются *расширения* последовательных языков *специальными конструкциями*, порождающими параллельные процессы (например, mpC, HPF, HPC++, Cilk, Maisie, MPL и др.); созданы языки *параллельного программирования* (Java, NESL, Orca, Sisal, ZPL и др.).

С другой стороны, можно использовать специальные *библиотеки* (например, PVM, MPI и т.д.), позволяющие организовывать асинхронный обмен сообщениями между потоками и процессами.

Организовать параллельную обработку можно, используя системные функции операционной системы, такие, например, как Portable Operating System Interface for UNIX (POSIX). Это стандарты, описывающие интерфейсы между ОС и прикладной программой. В пределах POSIX параллельные вычисления реализуются на основе обмена сообщениями (как в MPI) или разделяемой памяти (как в OpenMP).

Общий недостаток этих методов состоит в том, что очень большой объем работы по организации параллельной обработки возлагается на программиста. В связи с этим возникает задача минимизации ручной работы при проектировании параллельных программ и организации параллельной обработки данных.

Количество строк кода для параллельной программы значительно превышают объем кода для последовательного исполнения при программировании одной и той же задачи. Параллельный подход требует решения ряда дополнительных проблем, среди которых порождение и обработка синхронных событий, параллельные потоки, планирование и т.д. Если речь идет о многоядерных процессорах, то здесь могут использоваться различные модели многопроцессорности: асимметричная AMP для обеспечения управления и отказоустойчивости, симметричная SMP, поддерживающая масштабируемость и максимальный параллелизм, исключительная BMP, снижающая трудоемкость построения и миграцию кода.

В последнее время большие надежды для повышения эффективности решения задач, требующих большого объема вычислений, возлагают на Grid-системы, объединяющие огромное количество компьютеров, их программное обеспечение, средства коммуникаций и профессиональные человеческие ресурсы. Однако разработка и внедрение таких систем нуждается в решении ряда задач по координации работы и защите информации. Среди последних – проблемы аутентификации и авторизации, биллинга и аудита, строгого выполнения обязательств, конфиденциальности и целостности информации.

Организация параллельных вычислений предполагает разработку программной поддержки проектирования параллельных программ и организацию параллельной обработки данных.

Для решения первой части задачи, т.е. проектирования параллельных программ, введем конечное множество операторов T = {T_1, T_2, … T_n} и множество условий $U = \{u_1^i, u_2^i, ... u_k^i\}$. Оператор T_i осуществляет некоторое преобразование исходной программы P_o, проверяя серию условий U. При этом условие u_1^i является условием применимости оператора T_i к текущему состоянию программы P_0. Если условие применимости u_1^i не выполняется, то оператор T_i оставляет текущее состояние программы без изменения.

Для каждого оператора T_i задается S_i функций $f_1^i, f_2^i, ..., f_{S_i}^i$. Если оказывается, что оператор T_i применим к P_0, то проверяются последовательно условия $u_2^i, u_3^i, ... u_k^i$ и выполняются соответствующие преобразования программы, реализуя функции f_j^i. Завершив последовательную трансформацию P_0, мы получим ее параллельный функциональный эквивалент. Под функциональной эквивалентностью понимается следующее: при одном и том же входном наборе данных обе программы P_0 и P_n дают один и тот же результат.

Схематично процесс преобразования может быть представлен следующим образом:

$$P_0 \xrightarrow{T_1} P_1 \xrightarrow{T_2} P_2 \rightarrow ... \xrightarrow{T_n} P_n.$$

где P_0 – исходная программа, P_n – параллельная программа.

Укажем на две из возможных реализаций данной схемы.

При первой реализации можно предварительно разбить P_0 на фрагменты и вдоль программы расставить метки, специфицирующие каждый фрагмент. Тогда условие u_1^i определяет, может ли данный фрагмент быть преобразован оператором T_i. Если нет, то проверяется условие u_1^{i+1} на предмет применимости оператора T_{i+1} к данному фрагменту, и так до тех пор, пока не будет найден преобразующий его оператор. Если ни один из операторов T_i не применим к данному фрагменту, то последний остается без изменения и переходим к следующему.

Вторая реализация может быть следующей. Проверяется первый фрагмент программы на применимость оператора T_1, затем проверяется второй фрагмент, и так до тех пор, пока не будет обнаружен фрагмент, реализуемый оператором T_1. Таким же образом применяются остальные операторы.

Если проверка применимости оператора T_1 оказалась неуспешной, проверяется возможность применения оператора T_2, и так до последнего оператора.

Предложенная схема преобразования $P_0 \rightarrow P_n$ апробирована на одном классе последовательных программ и подтвердила целесообразность ее конкретизации, наполнения необходимыми процедурами, реализующими операторы T_i, и применения к проектированию параллельных программ.

Ясно, что особое внимание следует уделять эффективности реализации функций f_j^i, реализующих повторяющиеся фрагменты программы. Это объясняется тем, что ациклические участки в типичной ситуации выполняются лишь один раз, а фрагменты внутри циклов – многократно.

Организация параллельной обработки данных может быть решена посредством разбиения исходных данных на отдельные кванты. Кванты формируются таким образом, что они могут обрабатываться независимо. По завершению обработки идет интеграция полученного результата [3].

Предлагается процесс квантования базировать на перекодировке исходных данных в систему в коде вычетов (СКВ). В зависимости от свойств выбранной системы модулей алгоритм преобразования чисел из позиционной системы счисления (ПСС) в СКВ и обратно будет иметь различную сложность. В частности, если обобщить СКВ до безранговой СКВ (БСКВ), алгоритм преобразования ПСС $\overrightarrow{\leftarrow}$ СКВ будет проще алгоритма преобразования ПСС в двоичную систему и обратно.

Формула перевода из безранговой системы в ПСС будет иметь вид

$$A_{ПСС} = \sum_{i=1}^{n} x_i B_i,$$

где x_i – разряды (кванты) числа А в БСКВ в системе модулей $M_1, M_2, ..., M_n$, являющиеся наименьшими вычетами числа А по данным модулям, а j – ортогональные базисы, которые ищутся среди чисел вида $m_i \prod_{j=1}^{n} M_j$, где $i \neq j$, m_i – веса ортогональных базисов, принимающих значения 1, 2, …, M_i – 1.

Для анализа эффективности обработки данных предложенным способом выбраны три задачи: перемножение двух матриц, сортировка строк матрицы по возрастанию методом «пузырька», нахождение элемента матрицы с наибольшей суммой остатков от деления на элементы вектора. Задачи имеют одинаковую сложность вычислений, но различный удельный вес немодульных операций и допускают возможность изменения размерности задачи в процессе испытаний.

Результаты эксперимента приведены на рис. 2.

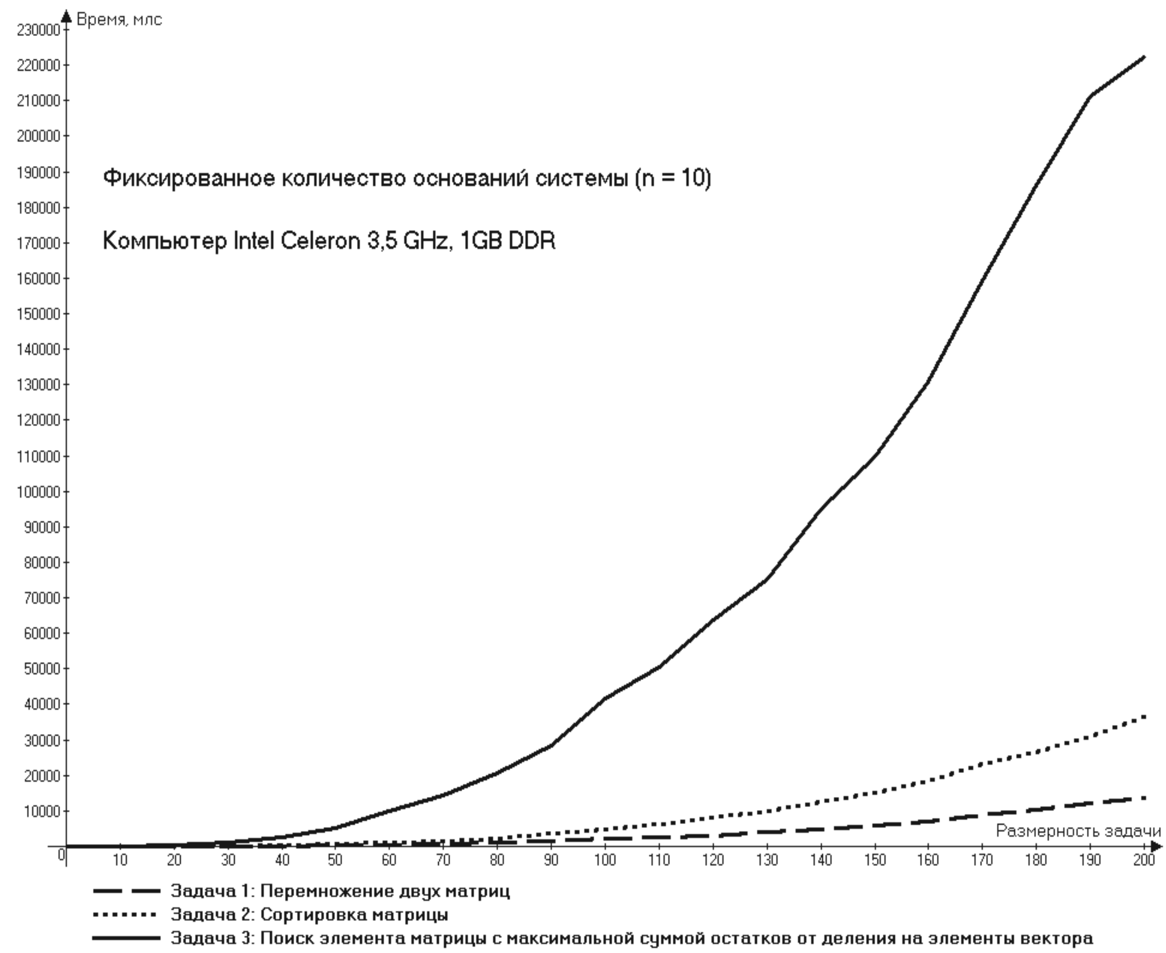


Рисунок 2 – Сравнение времени вычислений выбранных задач
в зависимости от их размерности

Как и предполагалось, наибольшее время обработки имеет задача, алгоритм решения которой содержит наибольшее количество немодульных операций. Исследования показывают, что максимальный эффект при использовании БСКВ получаем при обработке данных с небольшим количеством немодульных операций. Среди последних можно выделить операции сравнения и определения знака.

Заключение

Предложенная методика проектирования параллельной обработки является платформенно независимой, а качество ее применения в существенной степени зависит от эффективности реализации.

Для улучшения параметров обработки данных следует упрощать алгоритмы выполнения немодульных операций.

Разработанные программы могут быть использованы как при работе в компьютерной сети, так и в многопроцессорных системах. Они могут быть с успехом использованы и в мобильных центрах, производство которых наладила компания Sun Microsystems. Их система Project Blackbox смонтирована в обычном шестиметровом транспортном контейнере. В нем могут размещаться до 250 серверов, с общим объемом памяти до 7 Тбайт и емкостью дисковых накопителей более 2 Pбайт. Это вполне обеспечит обслуживание до 10^4 работающих на персональных компьютерах. Мобильный центр обойдется (по утверждению разработчиков) на два порядка дешевле традиционного компьютерного центра такой же мощности.

Литература

1. Троелсен Э. C# и платформа. NET. Библиотека программиста: Пер. с англ. – СПб.: Питер, 2007. – 800 с.
2. Барский А.Б. Параллельные информационные технологии. – М.: БИНОМ. – 2007. – 503 с.
3. Буза М.К. Архитектура компьютеров. – Минск: Новое знание, 2007. – 559 с.

М.К. Буза
Технологічний підхід до проектування паралельної обробки
У статті запропонований концептуальний підхід і методика проектування паралельних програм і метод прискореної обробки даних.

M.K. Buza
One Technology Way of Parallel Processing
Conceptual way, system of design parallel program and speed method of data processing are suggested.

Статья поступила в редакцию 24.06.2008.